HMMcopy: A package for bias-free copy number estimation and robust CNA detection in tumour samples from WGS HTS data

Daniel Lai and Gavin Ha

October 14, 2015

Contents

1 Introduction

High-throughput whole genome DNA sequencing results in millions of reads which after short-read mapping, can be used to infer the original sequence of the sample. An additional piece of information we gain from mapping is the coverage or number of reads overlapping each position, which can be used as a rough estimate of copy number. In practice, it is often easier to work at a lower resolution, and so we divide the genome into non-overlapping windows of fixed width (*e.g.* 1000), and use the *readcount* or number of reads starting in each of these windows (henceforth termed *bins*).

The main assumption of using readcount as a proxy for copy number is that during the sequencing protocol, reads are uniformly sampled from all genetic material present in the original input sample. This assumption has been shown to be incorrect, with reads being sampled at unequal rates depending on the GC content, among other things [1].

To different extents, this bias exists and varies in all platforms, protocols, and samples. Additionally, Benjamini and Speed [1] show that two libraries prepared from the same DNA source show different biases,

which highlights the fact that matched sample normalization (*i.e.* normalizing tumour copy by matched normal reference) is insufficient to eliminate this bias.

2 Generating Copy Number Profiles

For the remainder of this vignette, we will go through a short example using a matched normal tumour dataset, and show the ease and effects of normalization readcount by GC and mappability.

The dataset that is described here belongs to a female triple-negative breast cancer patient from the published dataset [2, 3]. This genome library was sequenced on the ABI/Life SOLiD platform, generating hybrid lengths of 25bp and 50bp paired-end reads. The reads were aligned using BioScope (`https://products.appliedbiosystems.com/`) where reads mapped to multiple sites were ignored.

2.1 Obtaining HTS copy number data

It should be noted that due to memory and speed considerations, the raw input that feeds in to this package are required to be generated by programs distributed as part of the *HMMcopy Suite* available at:

In short, the suite has tools to:

- Obtain high resolution bin counts for large ($\approx$ 250GB) BAM files within a few hours.
- Obtain GC content for bins from standard FASTA files within minutes for a human genome.
- Obtain average mappability for bins from BigWig within minutes, or FASTA files within a day for a human genome.

2.2 Loading HTS copy number data

To show the effects of correction in a simple case, let us begin by correcting data obtained from a normal genome sample.

To start, we load in WIG files containing three sets of data, namely readcounts, GC content, and average mappability, pre-computed for each bin with an external applications provided by the *HMMcopy Suite*.

Briefly, the readcounts data consists of the number of reads mapped within the boundaries of each bin. The GC content file consists of G and C base proportions for each bin; -1 is used when the reference sequence for the bin contains at least one unknown nucleotide base (i.e. N, rather than A,C, T or G). The mappability file, which was generated using *ENCODE Duke Uniqueness of 35bp sequences* track from UCSC (`http://genome.ucsc.edu/cgi-bin/hgTrackUi?db=hg18&g=wgEncodeMapability`), provides scores indicating how mappable the reference sequence at each bin is to aligners; the higher the score, the more mappable the sequence.

```
> library(HMMcopy)
> rfile <- system.file("extdata", "normal.wig", package = "HMMcopy")
> gfile <- system.file("extdata", "gc.wig", package = "HMMcopy")
> mfile <- system.file("extdata", "map.wig", package = "HMMcopy")
> normal_reads <- wigsToRangedData(rfile, gfile, mfile)
> normal_reads[1000:1010, ]

RangedData with 11 rows and 3 value columns across 1 space
      space               ranges |     reads        gc       map
   <factor>            <IRanges> | <integer> <numeric> <numeric>
1         6 [ 9990001, 10000000] |      7351    0.3932  0.558421
2         6 [10000001, 10010000] |     11080    0.3925  0.929000
3         6 [10010001, 10020000] |      9369    0.3689  0.975162
4         6 [10020001, 10030000] |      9505    0.3634  0.963870
```

```
5         6 [10030001, 10040000] |     10392    0.3903  0.889620
6         6 [10040001, 10050000] |     10785    0.3854  0.952784
7         6 [10050001, 10060000] |     11084    0.4264  0.916659
8         6 [10060001, 10070000] |      9712    0.3826  0.940946
9         6 [10070001, 10080000] |      5320    0.4011  0.422648
10        6 [10080001, 10090000] |     11014    0.3944  0.974912
11        6 [10090001, 10100000] |     10318    0.3667  0.971846
```

2.3 Correcting HTS copy number data

The resultant output is an IRanges object, which can easily use other Bioconductor functions and packages for analysis and visualzation. Although it is recommended that the output be left untouched until furthur correction as follows:

```
> normal_copy <- correctReadcount(normal_reads)
> normal_copy[1000:1010, ]
```

```
RangedData with 11 rows and 8 value columns across 1 space
      space               ranges |     reads        gc       map     valid
   <factor>            <IRanges> | <integer> <numeric> <numeric> <logical>
1         6 [ 9990001, 10000000] |      7351    0.3932  0.558421      TRUE
2         6 [10000001, 10010000] |     11080    0.3925  0.929000      TRUE
3         6 [10010001, 10020000] |      9369    0.3689  0.975162      TRUE
4         6 [10020001, 10030000] |      9505    0.3634  0.963870      TRUE
5         6 [10030001, 10040000] |     10392    0.3903  0.889620      TRUE
6         6 [10040001, 10050000] |     10785    0.3854  0.952784      TRUE
7         6 [10050001, 10060000] |     11084    0.4264  0.916659      TRUE
8         6 [10060001, 10070000] |      9712    0.3826  0.940946      TRUE
9         6 [10070001, 10080000] |      5320    0.4011  0.422648      TRUE
10        6 [10080001, 10090000] |     11014    0.3944  0.974912      TRUE
11        6 [10090001, 10100000] |     10318    0.3667  0.971846      TRUE
       ideal    cor.gc   cor.map         copy
   <logical> <numeric> <numeric>    <numeric>
1      FALSE 0.6667950 0.9462945 -0.079638885
2       TRUE 1.0083988 1.0391931  0.055463804
3       TRUE 0.9602569 0.9369282 -0.093989665
4       TRUE 1.0055782 0.9947256 -0.007629453
5      FALSE 0.9557880 1.0304725  0.043305954
6       TRUE 1.0154319 1.0179066  0.025605244
7       TRUE 0.8754294 0.9154566 -0.127436530
8       TRUE 0.9268861 0.9420144 -0.086178990
9      FALSE 0.4655253 0.7623204 -0.391530635
10      TRUE 0.9934066 0.9695664 -0.044588376
11      TRUE 1.0709252 1.0491263  0.069188320
```

```
>
```

The corrected reads are in the two columns `cor.gc` and `cor.map`, which are corrected and normalized, effectively a copy number estimation for each bin. For downstream analysis, `copy` is used, which is simply the base 2 log values of `cor.map`, to normalize the differences between values above and below 1. Specifically, the columns output from this process are:

valid Bins with valid GC and average mappability and non-zero read

ideal Valid bins of high mappability and reads that are not outliers

cor.gc Readcounts after the first GC correction step

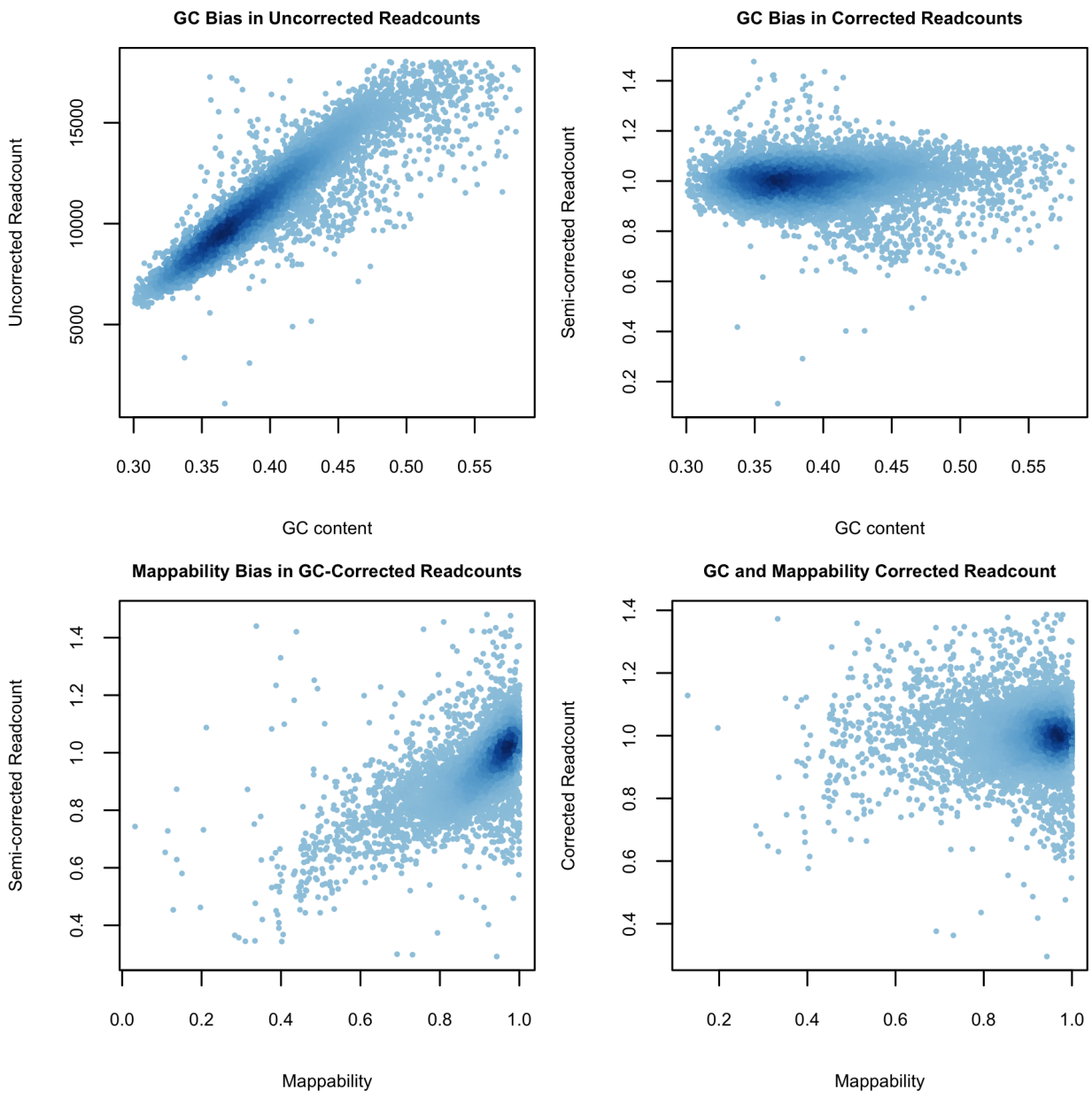
cor.map `cor.gc` readcounts after a furthur mappability correction

copy `cor.map` transformed into log2 space

2.4 Visualizing the effects of correction

To best visualize the biases in the original sample and the effects of correction, convenience plotting functions are available as follows:

```
> par(cex.main = 0.7, cex.lab = 0.7, cex.axis = 0.7, mar = c(4, 4, 2, 0.5))
> plotBias(normal_copy, pch = 20, cex = 0.5)
>
```

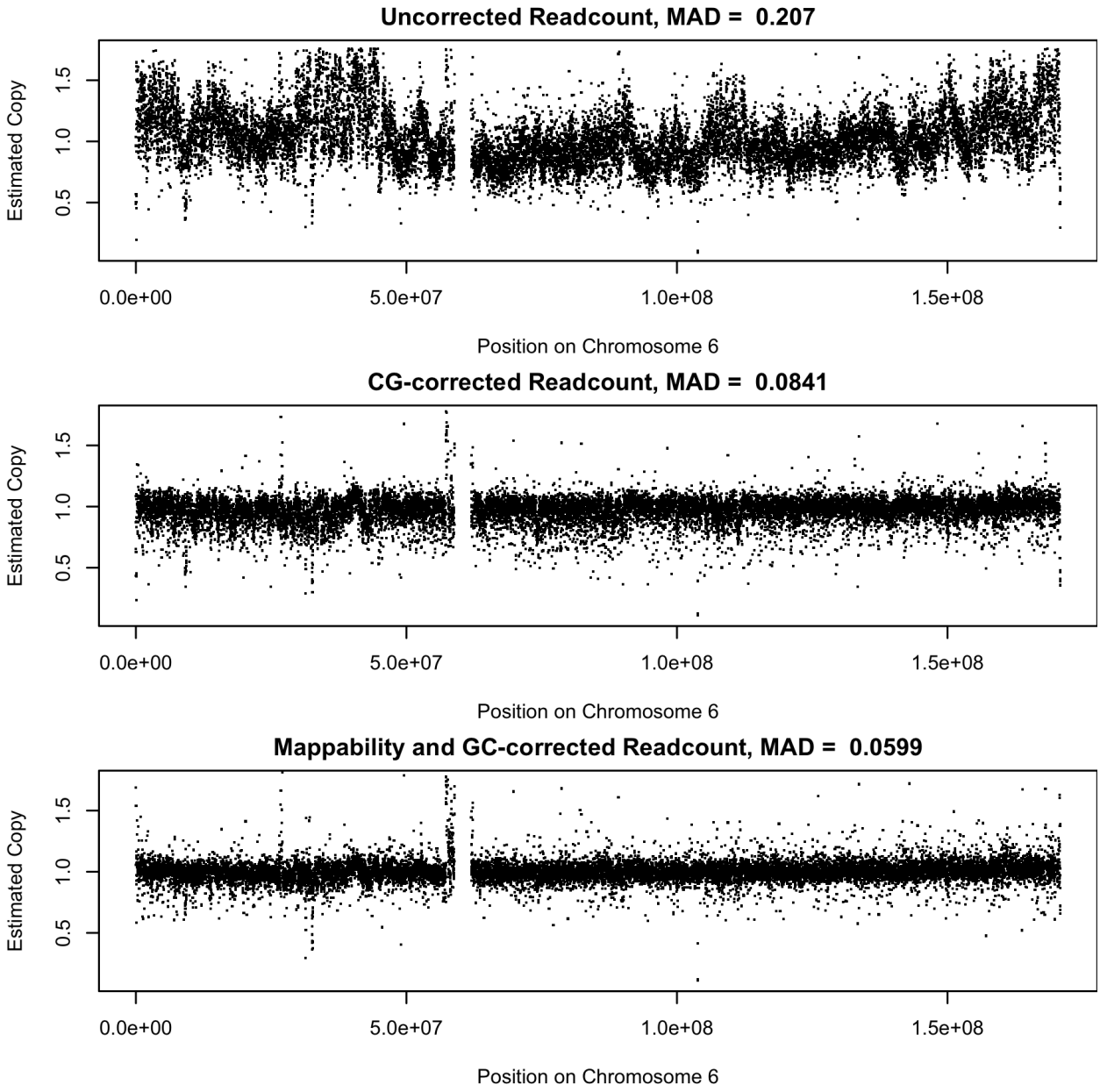


As observed, a unimodal relationship typically exists between readcount and GC content (excuse the weak tail), which is corrected and eliminated in the GC correction step. The GC-corrected readcounts exhibit a slight correlation with mappability, which is corrected in the second step.

2.5 Visualizing corrected copy number profiles

To see the effects of correction for copy number estimation, we can plot the copy number along a segment of a chromosome as follows:

```
> par(mar = c(4, 4, 2, 0))
> plotCorrection(normal_copy, pch = ".")
>
```

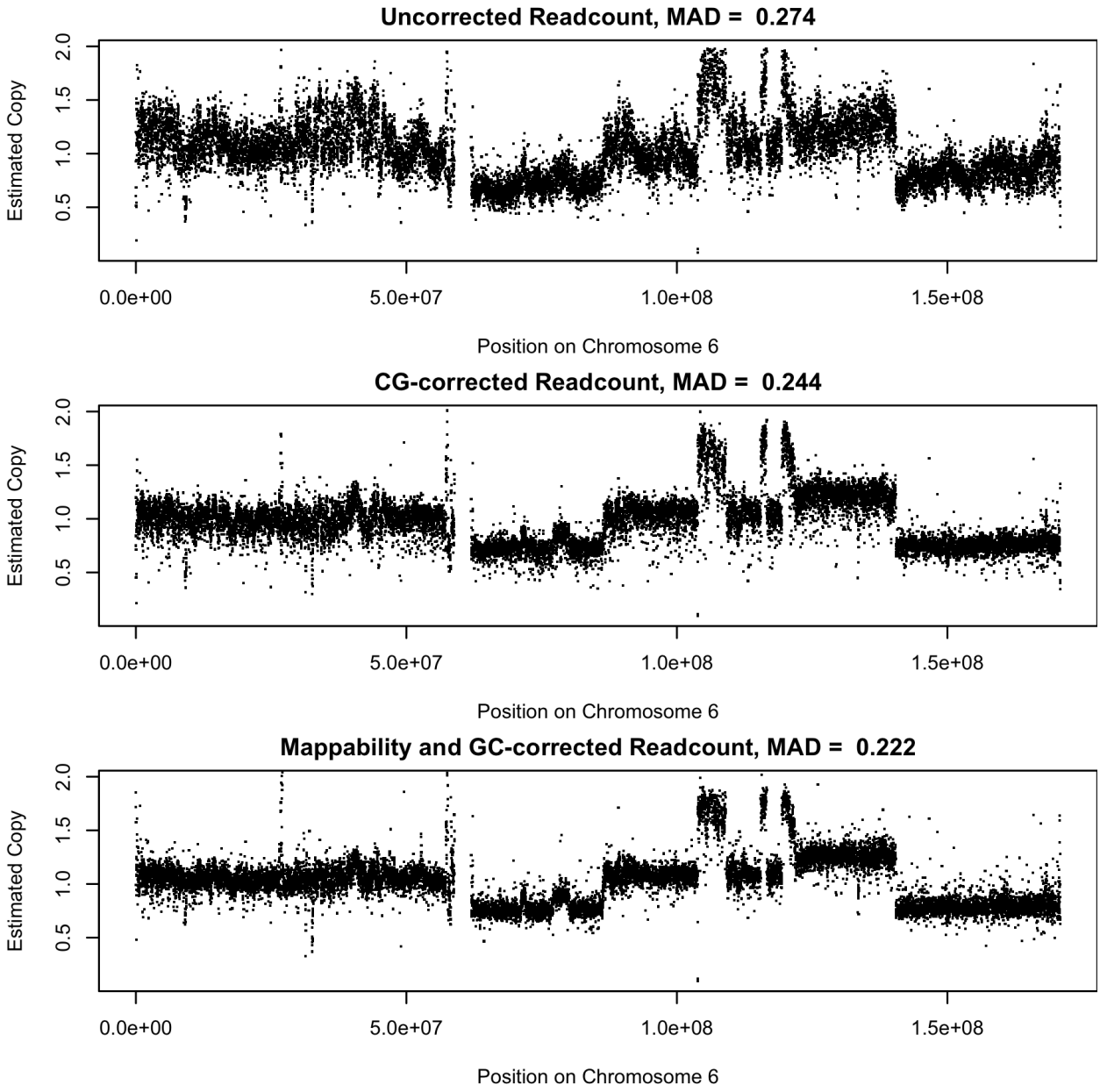


In the top track, the original uncorrected reads are used to estimate copy number (by simply dividing by the median value), followed by tracks with additional GC then mappability correction. Mediam absolute deviations (MAD) measuring the variance of each track can be seen to decrease after each correction step.

2.6 Correcting and visualizing tumour copy number profiles

The identical process can be applied to tumour genomes using the same GC and mappability files as follows:

```
> tfile <- system.file("extdata", "tumour.wig", package = "HMMcopy")
> tumour_copy <- correctReadcount(wigsToRangedData(tfile, gfile, mfile))
> par(mar = c(4, 4, 2, 0))
> plotCorrection(tumour_copy, pch = ".")
>
```



Note that at this stage, the correction of the normal and tumour samples were done *indepedently*, meaning that **HMMcopy is usable both in single sample and matched sample libraries.**

3 Segmentation and Classification of Copy Number Profiles

While corrected copy number data points are pleasing to stare at, they are often of much more practical use if we could group and classify them segments of specific copy number variation events. Included in the package is an implementation of an Hidden Markov Model [4] which does two main things:

1. Segments the copy number profile in regions predicted to be generated by the same copy number variation event

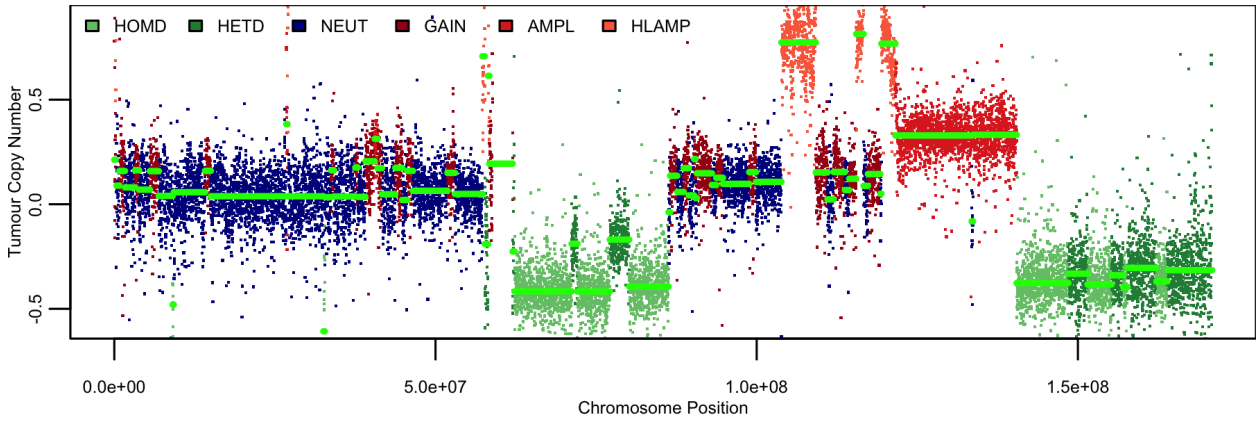
2. Predicts the copy number variation event for each segment, based on the biological likelihood of the event occuring

```
> tumour_segments <- HMMsegment(tumour_copy)
>
```

3.1 Visualizing segments and classified states

We can visualize both the resultant segments and the states assigned to each segment easily as follows:

```
> par(mfrow = c(1, 1))
> par(cex.main = 0.5, cex.lab = 0.5, cex.axis = 0.5, mar = c(2, 1.5, 0, 0), mgp = c(1, 0.5, 0))
> plotSegments(tumour_copy, tumour_segments, pch = ".",
+   ylab = "Tumour Copy Number", xlab = "Chromosome Position")
> cols <- stateCols() # 6 default state colours
> legend("topleft", c("HOMD", "HETD", "NEUT", "GAIN", "AMPL", "HLAMP"),
+   fill = cols, horiz = TRUE, bty = "n", cex = 0.5)
>
```



As observed, there are six coloured states, corresponding to six biological CNA events:

HOMD	Homozygous deletion, ≤ 0 copies
HETD	Heterozygous deletion, 1 copy
NEUT	Neutral change, 2 copies
GAIN	Gain of chromosome, 3 copies
AMPL	Amplification event, 4 copies
HLAMP	High level amplification, ≥ 5 copies

Segments are seen as neon green lines running through regions estimate to belong to the same CNA event.

3.2 Improving segmentation performance

The segmentation program will attempt its best at creating the most biologically likely state assignment, but unfortunately and fortunately for us humans, the automated program can often fail, as seen in this specific example. In cases like these, there is then a need to adjust the parameters generated by the algorithm. Instead of getting parameters from scratch however, one can easily retrieve the default set of parameters as follows:

```
> default_param <- HMMsegment(tumour_copy, getparam = TRUE)
> default_param

  strength         e          mu lambda  nu kappa           m   eta gamma
1    1e+07 0.9999999 -0.42054605     20 2.1    50 -0.42054605 5e+04     3
2    1e+07 0.9999999 -0.28184226     20 2.1    50 -0.28184226 5e+04     3
3    1e+07 0.9999999  0.04200362     20 2.1   700  0.04200362 5e+05     3
4    1e+07 0.9999999  0.18884920     20 2.1   100  0.18884920 5e+04     3
5    1e+07 0.9999999  0.36472889     20 2.1    50  0.36472889 5e+04     3
6    1e+07 0.9999999  0.89363465     20 2.1    50  0.89363465 5e+04     3
           S
1 0.01858295
2 0.01858295
3 0.01858295
4 0.01858295
5 0.01858295
6 0.01858295

>
```

By calling the same segmentation algorithm with the `getparam` argument set to `TRUE`, the algorithm generates parameters without actually running the segmentation and returns it. There are 10 parameters explained in detail in the documentation (*i.e.* `?HMMsegment`), each having values across six states in each row.

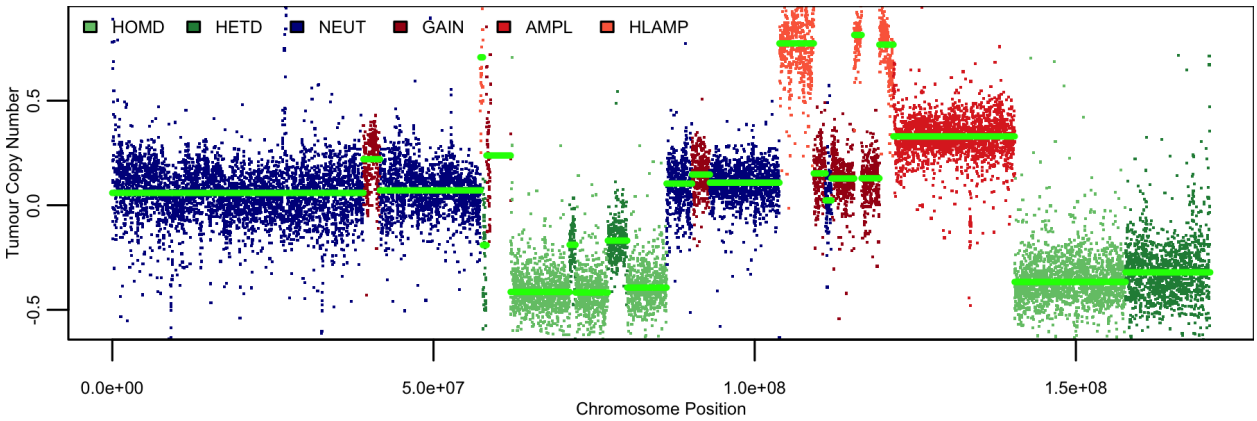
3.3 Reducing the number of segments

To reduce the segment, the two parameters we need to adjust are `e` and `strength`, which are the suggested probability of staying in a segment, and the strength of your suggestion to the algorithm. Once set, we run the segmentation algorithm, but this time expliciting providing our new parameters.

```
> longseg_param <- default_param
> longseg_param$e <- 0.999999999999999
> longseg_param$strength <- 1e30
> longseg_segments <- HMMsegment(tumour_copy, longseg_param, verbose = FALSE)
>
```

And finally, we can visualize the results to confirm a decrease in segments as intended.

```
> par(cex.main = 0.5, cex.lab = 0.5, cex.axis = 0.5, mar = c(2, 1.5, 0, 0), mgp = c(1, 0.5, 0))
> plotSegments(tumour_copy, longseg_segments, pch = ".",
+   ylab = "Tumour Copy Number", xlab = "Chromosome Position")
> legend("topleft", c("HOMD", "HETD", "NEUT", "GAIN", "AMPL", "HLAMP"),
+   fill = cols, horiz = TRUE, bty = "n", cex = 0.5)
>
```



As one can imagine, if for some odd reason you'd like to increase the number of segments, simply decrease `e` and `strength` to a small value (close to 0). Although the algorithm will almost certainly take a few times longer to run.

3.4 Adjusting copy number state ranges

A second more subtle and debilitating problem seen in this profile is actually the incorrect median of each copy number state. Specifically, this is a problem with the `mu` parameter. Accessing one of the output values of the segmentation process:

```
> longseg_segments$mus

            [,1]        [,2]        [,3]       [,4]        [,5]        [,6]
[1,] -0.42054605 -0.41758720 -0.41856737 -0.4189370 -0.41910605 -0.41919148
[2,] -0.28184226 -0.28218243 -0.28192091 -0.2819198 -0.28192063 -0.28192108
[3,]  0.04200362  0.04252202  0.04233246  0.0422898  0.04227474  0.04227086
[4,]  0.18884920  0.18763502  0.18755810  0.1877988  0.18806709  0.18818965
[5,]  0.36472889  0.36339188  0.36354889  0.3637438  0.36383827  0.36388573
[6,]  0.89363465  0.89183095  0.89189894  0.8919867  0.89203100  0.89205326
            [,7]
[1,] -0.41923759
[2,] -0.28192211
[3,]  0.04226854
[4,]  0.18825160
[5,]  0.36391057
[6,]  0.89206445

>
```

We see the median of 6 states (rows) after each iteration of the optimization algorithm (columns). The first column is our initial suggested `mu`:

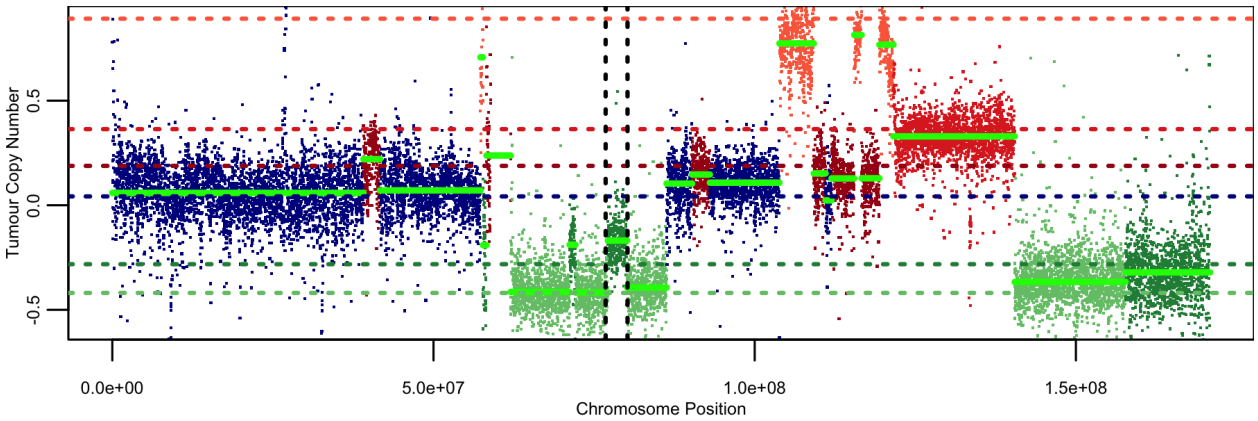
```
> longseg_param$mu

[1] -0.42054605 -0.28184226  0.04200362  0.18884920  0.36472889  0.89363465

>
```

And the last column is the actual values used during the segmentation process, which we can visualize as follows:

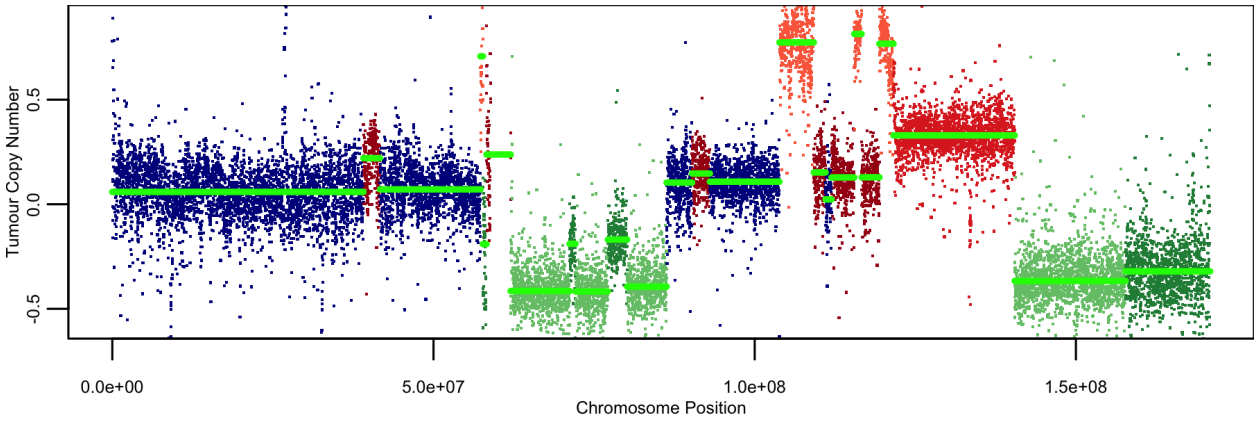
```
> par(cex.main = 0.5, cex.lab = 0.5, cex.axis = 0.5, mar = c(2, 1.5, 0, 0), mgp = c(1, 0.5, 0))
> plotSegments(tumour_copy, longseg_segments, pch = ".",
+   ylab = "Tumour Copy Number", xlab = "Chromosome Position")
> for(i in 1:nrow(longseg_segments$mus)) {
+   abline(h = longseg_segments$mus[i ,ncol(longseg_segments$mus)], col = cols[i],
+   lwd = 2, lty = 3)
+ }
> abline(v = 7.68e7, lwd = 2, lty = 3)
> abline(v = 8.02e7, lwd = 2, lty = 3)
>
```



From this, it becomes obvious that the medians are clearly not running through the middle of the visually obvious segments for many of the states. For example, the red medians for `GAIN`, `AMPL` and `HLAMP` are much too high, the green medians for `HOMD` and `HETD`are much too close together. Finally there is a clear segment flanked by vertical black dashed lines near the centre of the chromosome that is stuck between medians for green `HETD` and blue `NEUT`, when it should definitively belong to one or the other.

Staring at such a diagram, it is easy to suggest new `mu` params, which can be put back into the segmentation algorithm once again and visualized as follows.

```
> newmu_param <- longseg_param
> newmu_param$mu <- c(-0.5, -0.4, -0.15, 0.1, 0.4, 0.7)
> newmu_segments <- HMMsegment(tumour_copy, newmu_param, verbose = FALSE)
> par(cex.main = 0.5, cex.lab = 0.5, cex.axis = 0.5, mar = c(2, 1.5, 0, 0), mgp = c(1, 0.5, 0))
> plotSegments(tumour_copy, newmu_segments, pch = ".",
+   ylab = "Tumour Copy Number", xlab = "Chromosome Position")
>
```



3.4.1 Understanding parameter convergence

The observant reader will noticed nothing has changed. And this was done to purposely highlight that the parameters you set are often just *suggestions*, and given sufficient flexibility, the algorithm will effectively ignore your suggestions. As seen below in `newmu_segments`, after several iterations our newly suggested values of `mu` (first column) effectively converge (in the last column) to the same default values of `mu` in `longseg_param`.

```
> newmu_segments$mus
```

```
      [,1]        [,2]        [,3]        [,4]       [,5]        [,6]
[1,] -0.50 -0.42054605 -0.41943464 -0.41935974 -0.4193407 -0.41933805
[2,] -0.40 -0.29262755 -0.28206379 -0.28192587 -0.2819340 -0.28194784
[3,] -0.15  0.04183711  0.04199752  0.04218017  0.0423006  0.04228132
[4,]  0.10  0.17033461  0.17508748  0.18509160  0.1876240  0.18800732
[5,]  0.40  0.36329444  0.36360858  0.36377537  0.3638543  0.36389415
[6,]  0.70  0.89064389  0.89162704  0.89185993  0.8919678  0.89202060
            [,7]       [,8]
[1,] -0.41933936 -0.4193419
[2,] -0.28195775 -0.2819646
[3,]  0.04227209  0.0422691
[4,]  0.18814544  0.1882269
[5,]  0.36391512  0.3639264
[6,]  0.89204721  0.8920609
```

```
> longseg_param$mu
```

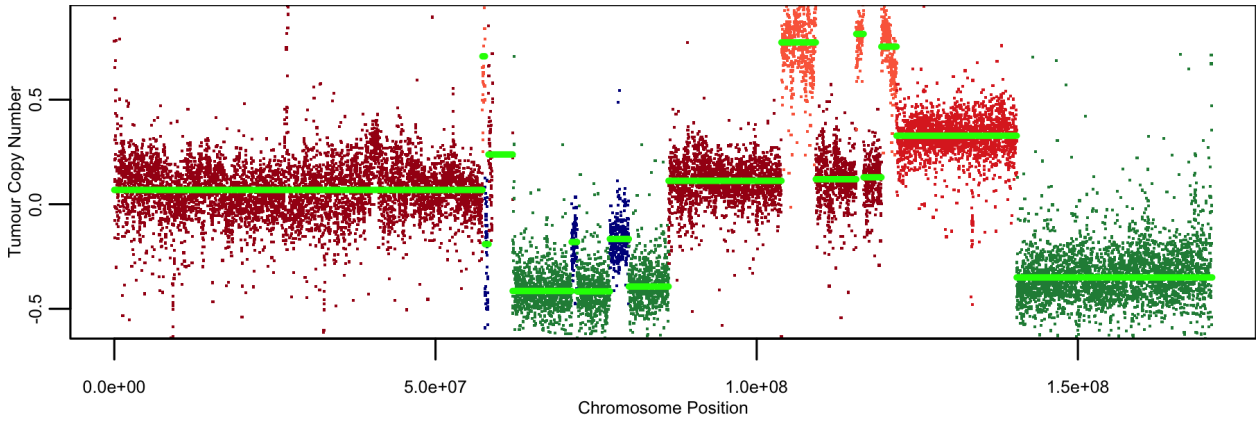
```
[1] -0.42054605 -0.28184226  0.04200362  0.18884920  0.36472889  0.89363465
```

```
>
```

3.4.2 Overriding parameter convergence

The solution is to simply disallow the algorithm from making large shifts to `mu`, which we can achieve by setting the prior mean of `n` (*i.e.* `m`) to values identical to `mu`.

```
> par(cex.main = 0.5, cex.lab = 0.5, cex.axis = 0.5, mar = c(2, 1.5, 0, 0), mgp = c(1, 0.5, 0))
> newmu_param$m <- newmu_param$mu
> realmu_segments <- HMMsegment(tumour_copy, newmu_param, verbose = FALSE)
> plotSegments(tumour_copy, realmu_segments, pch = ".",
+   ylab = "Tumour Copy Number", xlab = "Chromosome Position")
>
```



4 Matched Tumour-Normal Sample Correction

So far, our entire process has been done on single samples, providing respectable results and aesthetically pleasing profiles. However, when give matched tumour and normal pairs, we can improve our copy number results in two ways:

1. We can divide our tumour copy number profile by normal copy number profile to eliminate any germline copy number variation (CNV) events present in both normal and tumour profiles. This will leave only somatic copy number aberration (CNA) events in the tumour.

2. A dramatic reduction in noise is also observed, furthur improving copy number profiles.

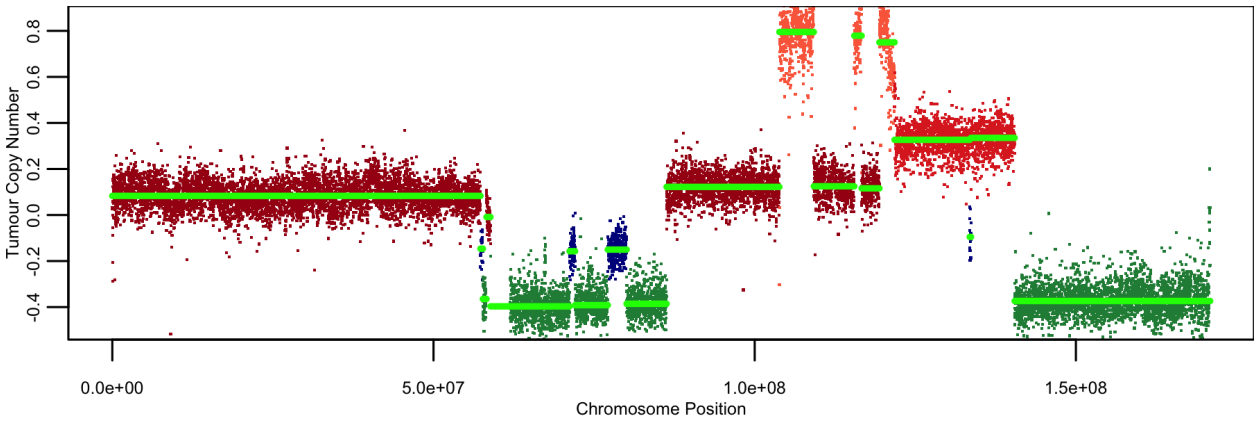
4.1 Normalizing tumour by normal copy number profiles

The normalization is a simple division of tumour copy number by normal copy number in a bin-wise fashion as follows, or a subtracting of the values in log space as follows:

```
> somatic_copy <- tumour_copy
> # LOGARITHM IDENTITY: log(a) - log(b) == lob(a / b)
> somatic_copy$copy <- tumour_copy$copy - normal_copy$copy
>
```

We can then do the same segmentation and visualization as follows:

```
> somatic_segments <- HMMsegment(somatic_copy, newmu_param, verbose = FALSE)
> par(cex.main = 0.5, cex.lab = 0.5, cex.axis = 0.5, mar = c(2, 1.5, 0, 0), mgp = c(1, 0.5, 0))
> plotSegments(somatic_copy, somatic_segments, pch = ".",
+   ylab = "Tumour Copy Number", xlab = "Chromosome Position")
>
```



The resultant corrected reads can easily be analyzed and exported with other R and Bioconductor packages.

5 Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 3.2.2 Patched (2015-10-08 r69496), `x86_64-apple-darwin10.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`

- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: annotate 1.48.0, AnnotationDbi 1.32.0, Biobase 2.30.0, BiocGenerics 0.16.0, geneplotter 1.48.0, HMMcopy 1.12.0, IRanges 2.4.0, lattice 0.20-33, S4Vectors 0.8.0, XML 3.98-1.3
- Loaded via a namespace (and not attached): DBI 0.3.1, grid 3.2.2, KernSmooth 2.23-15, RColorBrewer 1.1-2, RSQLite 1.0.0, tools 3.2.2, xtable 1.7-4

References

[1] Yuval Benjamini and Terence P Speed. Summarizing and correcting the gc content bias in high-throughput sequencing. *Nucleic Acids Res*, 40(10):e72, May 2012.

[2] Gavin Ha, Andrew Roth, Daniel Lai, Ali Bashashati, Jiarui Ding, Rodrigo Goya, Ryan Giuliany, Jamie Rosner, Arusha Oloumi, Karey Shumansky, Suet-Feung Chin, Gulisa Turashvili, Martin Hirst, Carlos Caldas, Marco A Marra, Samuel Aparicio, and Sohrab P Shah. Integrative analysis of genome-wide loss of heterozygosity and mono-allelic expression at nucleotide resolution reveals disrupted pathways in triple negative breast cancer. *Genome Research*, (advanced online publication), May 2012.

[3] S P Shah, A Roth, R Goya, A Oloumi, G Ha, Y Zhao, G Turashvili, J Ding, K Tse, G Haffari, A Bashashati, L M Prentice, J Khattra, A Burleigh, D Yap, V Bernard, A McPherson, K Shumansky, A Crisan, R Giuliany, A Heravi-Moussavi, J Rosner, D Lai, I Birol, R Varhol, A Tam, N Dhalla, T Zeng, K Ma, S K Chan, M Griffith, A Moradian, S W Cheng, G B Morin, P Watson, K Gelmon, S Chia, S F Chin, C Curtis, O M Rueda, P D Pharoah, S Damaraju, J Mackey, K Hoon, T Harkins, V Tadigotla, M Sigaroudinia, P Gascard, T Tlsty, J F Costello, I M Meyer, C J Eaves, W W Wasserman, S Jones, D Huntsman, M Hirst, C Caldas, M A Marra, and S Aparicio. The clonal and mutational evolution spectrum of primary triple-negative breast cancers. *Nature*, 486(7403):395–399, Jun 2012.

[4] Sohrab P Shah, Xiang Xuan, Ron J DeLeeuw, Mehrnoush Khojasteh, Wan L Lam, Raymond Ng, and Kevin P Murphy. Integrating copy number polymorphisms into array cgh analysis using a robust hmm. *Bioinformatics*, 22(14):e431–9, Jul 2006.